Курсовая работа по теме "Умножение разреженных матриц с уменьшенным количеством коммуникаций".

Егор Подлесов

20 мая 2024 г.

Оглавление

Введение 2
Методы подпространств Крылова 3
2.1 Структура Крыловских методов 3
2.2 Модель стоимостей вычислительных операций в Крыловских методах 3
Коммуникации при параллельных вычислениях 4
3.1 Что такое коммуникации в вычислительных алгоритмах 4
Снижение количества коммуникаций в умножении разреженных матриц в крыловских методах 5
Алгоритмы снижающие количество коммуникаций 6
5.1 Введение обозначений 6
5.2 Алгоритмы 6
Результаты 8
Заключение 9

Введение

Стоимость вычислительного алгоритма включает в себя как арифметику, так и обмен данными. Термин «обмен данными» используется в общем смысле для обозначения перемещения данных либо между уровнями иерархии памяти - «последовательный», либо между несколькими процессорами - «параллельный». Затраты на связь намного превышают арифметические затраты, и вместе с технологическим развитием разрыв быстро увеличивается. Это говорит о том, что для достижения наилучшей производительности алгоритмы должны минимизировать обмен данными, даже если для этого могут потребоваться некоторые избыточные арифметические операции. Такие алгоритмы называются «избегающими обменов».

Методы подпространств Крылова - это итерационные алгоритмы для решения больших разреженных линейных систем и задач на собственные значения. Современные методы подпространств Крылова тратят большую часть своего времени на такие операции, как умножение разреженной матрицы на вектор (SpMV) и векторно-векторные операции (включая скалярные произведения и суммы векторов). Кроме того, зависимости данных между этими операциями означают, что только небольшая часть этих коммуникаций может быть скрыта. Многие важные научные и инженерные вычисления выполняются в основном с использованием методов Крылова, поэтому производительность многих кодов можно было бы повысить, внедрив эти методы, обеспечивающие меньшее количество обменов.

Наша цель - выполнить s итераций метода с тем же количеством коммуникаций что было за 1 итерацию. Это можно сделать при определенных предположениях о матрице и для определенных методов подпространств Крылова, в теории для многих методов такого типа, на практике для некоторых. **Эта работа объясняет, как избежать коммуникаций в этих методах Крылова на операции SpMV**.

Методы подпространств Крылова

2.1 Структура Крыловских методов

Методы подпространства Крылова - это большой класс итерационных алгоритмов, которые работают с конечными матрицами и векторами в вещественной и сложной арифметике. Они имеют множество применений, включая решение линейных систем, задач наименьших квадратов, задач на собственные значения и задач с сингулярными значениями. В этой работе обсуждается решение линейных систем $Ax = b$ и задач на собственные значения $Ax = \lambda x$. В данной работе представляются методы Крылова, избегающие обменов.

Методы подпространства Крылова работают, используя одно или несколько матрично-векторных произведений на каждой итерации для добавления вектора (ов) к базису для одного или нескольких так называемых «подпространств Крылова» с желаемыми свойствами. Подпространство Крылова относительно квадратной матрицы $A_{n\times n}$ и вектора v длины n является следующим подпространством:

$$K_k(A, v) = \{\, v,\, Av,\, A^2v,\, \ldots,\, A^{k-1}v \,\}$$

где k - целое положительное число. В методах подпространств Крылова используется одно или несколько подпространств Крылова. Предполагается, что каждая итерация метода увеличивает размерность этих подпространств. Они используются в качестве расширяющегося пространства, из которого с помощью проекции вычисляется приближенное решение x_k (если решается $Ax = b$) или приближенные собственные значения и векторы (если решается $Ax = x$). Ниже речь пойдет о конкретном методе споряженных градиентов.

2.2 Модель стоимостей вычислительных операций в Крыловских методах

Каждая итерация метода сопряженных градиентов может состоять из

1. $SpMV(A, b)$ - умножение разреженной матрицы $A_{n\times n}$ на вектор v длины n.
2. $InnerProduct(a, b)$ - скалярное произведение векторов a и b.
3. $AXPY(\beta, a, \alpha, y)$ - линейная комбинация векторов $a = \beta \cdot a + \alpha \cdot y$.

Самая дорогая с точки зрения коммуникаций операция - SpMV.

Коммуникации при параллельных вычислениях

3.1 Что такое коммуникации в вычислительных алгоритмах

Если мы хотим избежать коммуникаций, мы сначала должны понять, что такое «коммуникации». В этой работе предполагается, что компьютеры при исполнении программы выполняют: арифметические вычисления и коммуницируют. Коммуникация определяется как перемещение данных. Это включает в себя как перемещение данных между уровнями иерархии памяти, которое мы называем последовательной передачей, так и перемещение данных между процессорами, работающими параллельно, которое мы называем параллельной передачей. В иерархию памяти включаются все формы хранения, подключенные к центральному процессору (CPU), такие как кэш, основная память, твердотельный накопитель, жесткий диск или ленточный архив. Они могут быть подключены непосредственно к центральному процессору или через сетевой интерфейс. Основная память современных компьютеров, как правило, состоит из DRAM (динамической оперативной памяти), поэтому мы используем DRAM как синоним основной памяти.

Взаимодействие между параллельными процессорами может принимать, среди прочего, следующие формы:

- Обмен сообщениями между процессорами в системе с распределенной памятью
- Контроль согласованности кэширования в системе с общей памятью
- Передача данных между сопроцессорами, соединенными шиной, например, между центральным процессором и графическим процессором («Графический процессор»)

Если мы рассматриваем два уровня в иерархии памяти, мы называем их «быстрой» памятью и «медленной» памятью. Быстрая память - это уровень с меньшей задержкой для процессора и, как правило, меньшей емкостью, чем медленная память. «Быстрый» и «медленный» всегда определяются в сравнении. Например, при рассмотрении кэша и DRAM кэш - это «быстрый», а DRAM - «медленный». Однако, если рассматривать DRAM и диск, то DRAM - это «быстрый», а диск - «медленный».

Снижение количества коммуникаций в умножении разреженных матриц в крыловских методах

Алгоритм умножения разреженной матрицы на вектор с уменьшенным числом коммуникаций называется «MatrixPowers kernel». Будем называть его «вычислительным ядром». Ядро MatrixPowers берет разреженную матрицу A и плотный вектор v и вычисляет s векторов Av, A^2v, ..., A^sv. Для большого класса разреженных матриц ядро MatrixPowers может вычислить эти векторы для оптимально с точки зрения количества коммуникаций.

В последовательном случае «минимальный» означает, что и матрица A, и векторы v, Av, ..., A^sv нужно перемещать между быстрой и медленной памятью всего $1+o(1)$ раз. Наивная реализация этого ядра с использованием s последовательных вызовов SpMV потребовала бы считывания разреженной матрицы A из медленной памяти s раз.

В параллельном случае «минимальное» означает, что требуемое количество сообщений на процессор может быть уменьшено в $\Theta(s)$ раз, так что количество сообщений для вычислительного ядра MatrixPowers будет таким же, как для $1 + o(1)$ вызовов параллельного SpMV. Поскольку многие стандартные методы подпространства Крылова тратят большую часть времени выполнения на SpMV, замена операций SpMV вычислительным ядром MatrixPowers может значительно ускорить выполнение.

Алгоритмы снижающие количество коммуникаций

5.1 Введение обозначений

Дана разреженная матрца A размера $n \times n$. С ней ассоциируется направленный граф G такой, что если $A_{ij} \neq 0$, то есть ребро из вершины i в вершину j. В контексте вычисления матрично векторного произведения $y = A \cdot x$ это значит, что значение y_i зависит от значения x_j.

Чтобы снизить количество коммуникаций в нашей процедуре MatrixPowers, требуется установить зависимости между матрично-векторными умножениями. Например такое множество j, что для $z = A^k x$ следует, что z_i зависит от x_j. Построим граф, выражающий такие зависимости.

Пусть $x_j^{(i)}$ это j-ая компонента вектора $x^{(i)} = A^i \cdot x^{(0)}$. Каждому $x_j^{(i)}$ для всех $i \in \overline{0,s}$ и $j \in \overline{1,n}$ соответствует одноименная вершина.

Если $A_{jm} \neq 0$, то из вершины $x_j^{(i+1)}$ в $x_m^{(i)}$ есть ребро.

Вышеописанный граф из $n \cdot (s+1)$ вершин обозначим G.

Назначим каждой вершине некоторый параметр «степень локальности» q. В сущности он обозначает разное для случаев параллельных и последовательных алгоритмов.

G_q - множество вершин графа с степенью локальности равной q.

$G_q^{(i)}$ - множество вершин графа с степенью локальности равной q и имеющих уровень i.

Пусть S - некоторое множество вершин графа G. Обозначим $R(S)$ - множество вершин в которые есть путь в графе начинающийся из вершины множества S. Аналогично вводятся обозначения $R(S)_q$ и $R(S)_q^{(i)}$.

Введем также множество L_q. Определим его следующим образом:

$$L_q = \{x \in G_q \ : \ R(x) \subset G_q\}$$

Аналогично вводится множество $L_q^{(i)}$.

Введём множество вершин $B_{q,r}$. $x \in B_{q,r}$, тогда и только тогда, когда $x \in L_r$ и существует такой путь из некоторой вершины $y \in G_q$ в x, что x - первая вершина в этом пути из множества L_r.

5.2 Алгоритмы

Algorithm 1 PA1 Алгоритм (Код для процессора q)

1: **for** $i = 1$ to s **do**
2: **for all** processors $r \neq q$ **do**
3: Send all $x_j^{(i-1)}$ in $R_q^{(i-1)}(G_r^{(i)})$ to processor r
4: **for all** processors $r \neq q$ **do**
5: Receive all $x_j^{(i-1)}$ in $R_r^{(i-1)}(G_q^{(i)})$ from processor r
6: Compute all $x_j^{(i)}$ in $L_q^{(i)}$
7: Wait for above receives to finish
8: Compute remaining $x_j^{(i)}$ in $G_q^{(i)} \setminus L_q^{(i)}$

Algorithm 2 PA2 Алгоритм (Код для процессора q)

1: **for all** processors $r \neq q$ **do**
2: Send all $x_j^{(0)}$ in $R_q^{(0)}(G_r)$ to processor r
3: **for all** processors $r \neq q$ **do**
4: Receive all $x_j^{(0)}$ in $R_r^{(0)}(G_q)$ from processor r
5: **for** $i = 1$ to s **do**
6: Compute all $x_j^{(i)}$ in L_q
7: Wait for above receives to finish
8: **for** $i = 1$ to s **do**
9: Compute remaining $x_j^{(i)}$ in $R(G_q) \setminus L_q$

Algorithm 3 PA3 algorithm (Code for processor q)

1: **for** $i = 1$ to s **do**
2: Compute $x_j^{(i)}$ in $\bigcup_{r \neq q}(R(Gr) \setminus L_q)$
3: **for all** processors $r \neq q$ **do**
4: Send $x_j^{(i)}$ in $B_{r,q}$ to processor r
5: **for all** processors $r \neq q$ **do**
6: Receive $x_j^{(i)}$ in $B_{r,q}$ from processor r
7: **for** $i = 1$ to s **do**
8: Compute $x_j^{(i)}$ in $L_q \setminus \bigcup_{r \neq q}(R(Gr) \setminus L_q)$
9: Wait for above receives to finish
10: **for** $i = 1$ to s **do**
11: Compute remaining $x_j^{(i)}$ in $R(G_q) \setminus L_q \setminus \bigcup_{r \neq q}(R(G_q) \setminus L_r)$

Результаты

Реализацию «MatrixPowers kernel» на C++ с использованием std::threads можно увидеть на гитхаб. Результаты бенчмаркинга можно увидеть там же. На размерностях до 2048 наша реализация или проигрывает или не существенно превосходит последовательное применение матрично-векторных умножений. Но начиная с размерности 4096 выигрыш в 3 раза становится существенным и увеличивается с ростом размерности.

Краткие результаты можно видеть ниже.

Method	**Size**	**Time**
MatrixPowersMV	1024	286.653 ms
Sequential MatVec's	1024	272.036 ms
MatrixPowersMV	2048	1.9255 s
Sequential MatVec's	2048	1.92448 s
MatrixPowersMV	4096	5.37807 s
Sequential MatVec's	4096	7.90444 s
MatrixPowersMV	8192	6.71484 s
Sequential MatVec's	8192	19.241 s
MatrixPowersMV	16384	23.5549 s
Sequential MatVec's	16384	1.2459 s

Заключение

В дальнейшие планы входит реализация данного вычислительного метода с использованием **MPI**, а также рассмотрение выигрыша на крыловских методах.